线性dp

2019年2月27日 17:36

-
- Dynamic Programming

一. 动态规划

1. 动规三要素
   i. 阶段：求解每个子问题的过程
   ii. 状态：有向无环图中的各节点
   iii. 决策：在有向无环图的各节点间的转移
2. 能用动规求解的三个基本条件
   i. 子问题重叠性：能将原问题视作若干重叠的子问题，逐层递进
   ii. 无后效性：已求解的子问题不受后续阶段的影响
   iii. 最优子结构性质：下一阶段的最优解能由各阶段子问题的最优解导出

二. 数字三角矩阵

1. 问题：i行i列的矩阵a，左上角出发，每次向下方或右下方走一步，并加上该位置的数，到最下一行的最大和为多少
2. 阶段ij：顺序遍历到当前点的行下标和列下标
3. 状态ans[i][j]：到当前点的最大和
4. 转移方程：ans[i][j]=a[i][j]+max(ans[i-1][j-1],ans[i-1][j])（正上或左上）
5. 初值：左上角ans=左上角a即可（不需要保留a的话，似乎都不需要开ans）
6. 答案：最下一行的最大值
   -
   - 例题

一. 以两人或三人为一组，n人的分组可能数

1. 分析：最后一组要么是2人，要么是3人，所以n人的组法取决于n-2人的组法和n-3人的组法之和
2. 初值：ans[0]=ans[1]=0; ans[2]=ans[3]=1;
3. for__(i,4,n)
   ans[i]=ans[i-2]+ans[i-3];

二. 过河，每个位置有一个弹簧，对应一个该点起跳最远距离t，求最少跳几次

1. 分析：用数组存储跳到每个点需要的最少步数，顺序循环到终点即可；每读入一个t，就判断一次后t个点是不是可以通过该点更快的到达
2. 转移方程：因为是顺序读入，能跳到当前点之后的点的话，一定也能跳到当前点，所以从当前点跳是不可能比早已能跳到的点更快的，能更新的只有之前不能跳到的点，更新值一定是当前点的次数+1
3. 初值：memset(cnt,-1,n*sizeof(int)); //初值为届不到
4. for_(i,0,n){
   scanf("%d",&t);
   if(t==0&&cnt[i+1]==-1){ //有不连通点，提前失败
   printf("-1");

```
            return 0;
        }else if(n-i<=t){   //一步入魂，提前成功
            printf("%d",cnt[i]);
            return 0;}
        int newc=cnt[i]+1;
        for__(j,1,t){
            if(cnt[i+j]==-1)  //更新
                cnt[i+j]=newc;}}
    printf("%d",cnt[n-1]);
```

三. 每秒随机在0~10掉几个馅饼，求起点在5，每秒能移一步的人最多能接到几个饼

1. 因为起点框定在了5，所以必须倒着走
2. 转移方程是后一秒三个点的最大值，不过左右端点只有两个
3.
```
rof__(i,T-1,0){
    a[0][i]+=max(a[0][i+1],a[1][i+1]);
    a[10][i]+=max(a[10][i+1],a[9][i+1]);
    for_(j,1,10)
        a[j][i]+=max(a[j][i+1],max(a[j-1][i+1],a[j+1][i+1]));}
cout<<a[5][0];
```

四. m次移动后t洞内出现地鼠的方案数，总共n洞，起点s洞，每天必跑去相邻任一洞

1.
```
const int MN= 10005;
int p= 19260817;
int n,m,s,t;
int d[MN][MN];
inline bool lgl(int x){
    return x>1 && x<n;
}
```
2.
```
cin>>n>>m>>s>>t;
++d[0][s];
for__(i,1,m){
    for__(j,s-i,s+i)
        if(lgl(j))
            d[i][j]= (d[i-1][j-1] + d[i-1][j+1])% p;
        else if(j==1)
            d[i][1]= d[i-1][2];
        else if(j==n)
            d[i][n]= d[i-1][n-1];}
cout<<d[m][t];
```

五. 安排合影站位的方法数，使每排从左到右，每列从上到下身高递增

1. 分析：按身高从矮到高的顺序遍历每个学生就可以避免复杂的排序，此时每次安排学生需要考虑当前各行还剩多少人
2. 阶段i：遍历每个学生
3. 阶段j：每行有多少空位
4. 状态ans[j1][j2][j3][j4][j5]……，每行的已安排人数分别为j1,j2……时有几种方法
5. 转移：ans[j1][j2]……的第一行还有空位时，给ans[j1+1][j2]的状态+=当前状态ans[j1][j2]……，同理第二行，第三行……